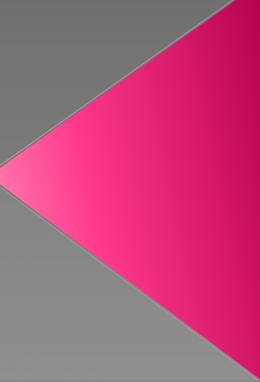




Elixir

Luiz Augusto Penna e Fábio Pimentel



Origens e Influências

- **Criador:** José Valim
- **Objetivos:** Permitir maior produtividade e extensibilidade no Erlang VM, mantendo a compatibilidade com ferramentas e ecossistemas de Erlang.
- **Data de criação:** 2012
- **Influenciada por:** Erlang

Classificação

- **Funcional:** Functions e Modules, não existem objetos ou classes.
- **Dinâmica:** Tipos são checados na execução
- **Compilada:** Código fonte são transformados em bytecodes que rodam sobre a Erlang VM.
- **Concorrência:** Usando modelo de Actors

Concorrência e Paralelismo

- Na programação concorrente e paralela um programa pode ser executado através de diversas linhas de execução.
- Ganho de desempenho em sistemas com muitos processadores (paralelismo físico)
- Ganhos de desempenho através de acesso concorrente a dispositivos de hardware

Operador PIPE(|>)

```
foo(bar(baz(nova_funcao(outra_funcao())))))
```



```
outra_funcao() |> nova_funcao() |> baz() |> bar() |> foo()
```

Operador PIPE(| >)

Exemplo

```
defmodule Calculadora do
    def imprime_soma(range) do
        IO.puts Enum.sum(range)
    end

    def imprime_soma_pipe(range) do
        range |> Enum.sum |> IO.puts
    end
end

Calculadora.imprime_soma(1..10)
Calculadora.imprime_soma_pipe(1..10)
```

Conceitos de Pattern Matching

- Pattern matching é uma poderosa parte de Elixir, nos permite procurar padrões simples em valores, estruturas de dados, e até funções.
- Em Elixir, o operador `=` é na verdade o nosso operador **match**. Quando usado, a expressão inteira se torna uma **equação** e faz com que Elixir o combine os valores do lado esquerdo com os valores do lado direito da expressão.

Exemplo de Pattern Matching

- A linguagem vai sempre tentar "casar" o valor da esquerda com o valor da direita.

```
iex(1)> a = 1
1
iex(2)> 1 = a
1
iex(3)> 2 = a
** (MatchError) no match of right hand side value: 1
```

Exemplo de Pattern Matching

```
defmodule Calculadora do
  def faz_operacao(a,b, tipo) do
    if tipo == :soma do
      a + b
    else
      a - b
    end
  end
end

Calculadora.faz_operacao(20,10, :soma) |> IO.puts
Calculadora.faz_operacao(20, 10, :subtracao) |> IO.puts
```

Exemplo de Pattern Matching

```
defmodule Calculadora do
  def faz_operacao(a,b, :soma) do
    a + b
  end

  def faz_operacao(a,b, :subtracao) do
    a-b
  end

end
Calculadora.faz_operacao(20,10, :soma) |> IO.puts
Calculadora.faz_operacao(20, 10, :subtracao) |> IO. puts
```

Exemplo de Pattern Matching

```
defmodule Calculadora do
    def fibonnacci(0) do
        0
    end

    def fibonnacci(1) do
        1
    end

    def fibonnacci(n) do
        fibonnacci(n-1)+ fibonnacci(n-2)
    end

end

2 |> Calculadora.fibonnacci |> IO.puts
5 |> Calculadora.fibonnacci |> IO.puts
11|> Calculadora.fibonnacci |> IO.puts
```

Exemplo de Pattern Matching

```
defmodule Calculadora do

    def fibonnacci(0), do: 0
    def fibonnacci(1), do: 1
    def fibonnacci(n), do: fibonnacci(n-1)+ fibonnacci(n-2)

end

2 |> Calculadora.fibonnacci |> IO.puts
5 |> Calculadora.fibonnacci |> IO.puts
11 |> Calculadora.fibonnacci |> IO.puts
```

Conceitos de Metaprogramação

A Metaprogramação é um recurso muito comum de linguagens dinâmicas permite a utilização de códigos para escrever código.

Em outras palavras podemos dizer que é a capacidade de gerar/alterar código em tempo de execução.

Conceitos de Metaprogramação

- Para entender esse conceito é necessário a compreensão de como as expressões são representadas. Em Elixir, **a árvore de sintaxe abstrata(AST)**, que é a representação interna do nosso código, é composta de tuplas. Essas tuplas contêm três partes:
 - Nome da função
 - Metadados
 - Argumentos da função

```
#tupla para AST
{function_call, meta_data_for_context, argument_list}
```

Conceitos de Metaprogramação

- Quote

```
iex(1)> quote do: 1+2
{:+, [context: Elixir, import: Kernel], [1, 2]}
```

Conceitos de Metaprogramação

- Quote em literais:

1) Número(floats e integers):

```
iex(5)> quote do: 1  
1
```

2) Átomo:

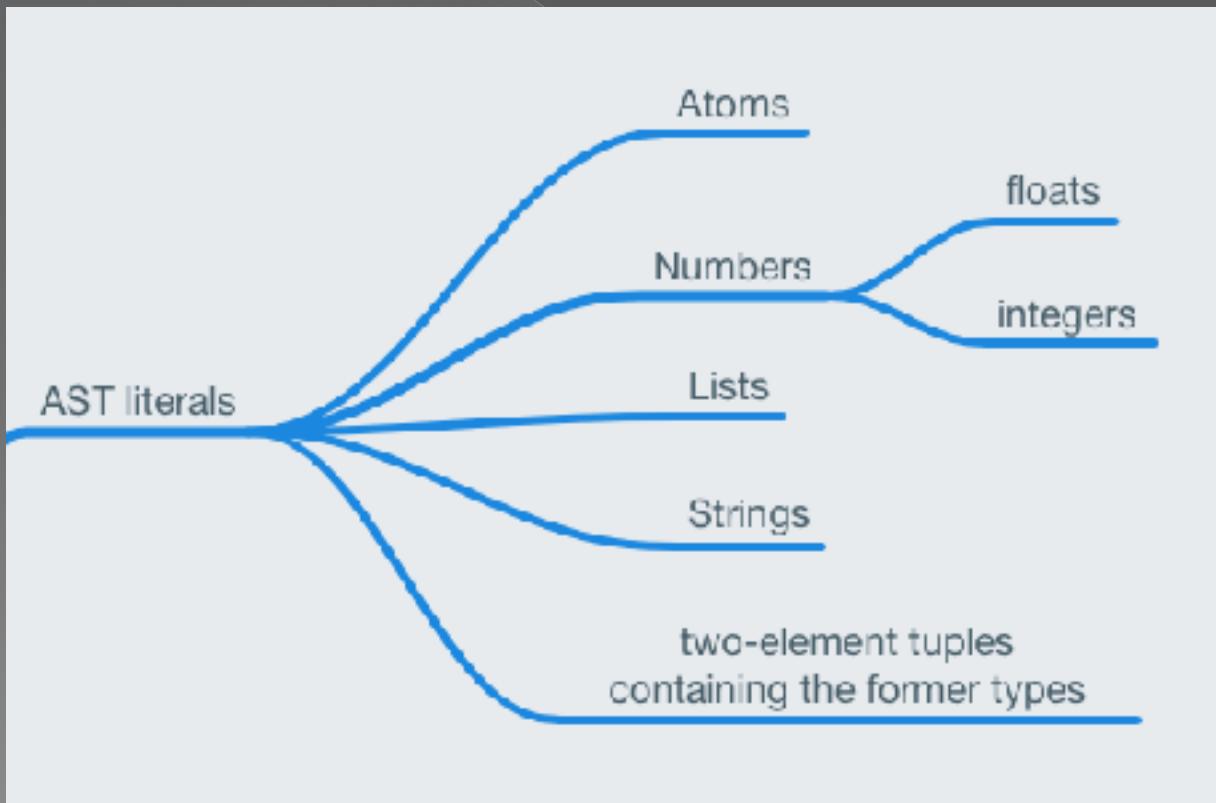
```
iex(6)> quote do: :adicao  
:adicao
```

3) List

```
iex(7)> quote do: [1,2,3]  
[1, 2, 3]
```

Conceitos de Metaprogramação

- Quote em literais:



Conceitos de Metaprogramação

- Unquote

```
iex(19)> operando = 10  
10
```

```
iex(20)> quote do: 20 + operando  
{:+, [context: Elixir, import: Kernel], [20, {:operando, [], Elixir}]}
```

```
iex(21)> quote do: 20 + unquote(operando)  
{:+, [context: Elixir, import: Kernel], [20, 10]}
```

Macros - onde a
metaprogramação ocorre...

Exemplo de Metaprogramação

```
Calculadora.debug(1+2)  
Calculadora.debug(2*5)
```

Exemplo de Metaprogramação

```
Calculadora.debug(1+2)
Calculadora.debug(2*5)
```

Resultado de 1+2: 3

3

Resultado de 2*5: 10

10

Exemplo de Metaprogramação

```
Calculadora.debug(1+2)
Calculadora.debug(2*5)
```

```
Resultado de 1+2: 3
3
Resultado de 2*5: 10
10
```

```
resultado = 1+2
Calculadora.print("1+2", resultado)
resultado
```

Exemplo de Metaprogramação

```
defmodule Calculadora do
  defmacro debug(expressao_quote) do
    quote do
      resultado = unquote(expressao_quote)
      Calculadora.print("1-2", resultado)
      resultado
    end
  end

  def print(string_representation, result) do
    IO.puts "Resultado de #{string_representation}: #{result}"
  end
end
```

Exemplo de Metaprogramação

```
defmodule Calculadora do
  defmacro debug(expressao_quote) do
    expressao_string = Macro.to_string(expressao_quote)

    quote do
      resultado = unquote(expressao_quote)
      Calculadora.print(unquote(expressao_string), resultado)
      resultado
    end
  end

  def print(string_representation, result) do
    IO.puts "Resultado de #{string_representation}: #{result}"
  end
end
```

Exemplo de Metaprogramação

```
defmodule Main do
  require Calculadora

  def test do
    Calculadora.debug(1+2)
    Calculadora.debug(1*2)
    Calculadora.debug(5-1)
  end
end

Main.test
```

Exemplo de Metaprogramação

```
$ elixir teste7.exs
Resultado de 1 + 2: 3
Resultado de 1 * 2: 2
Resultado de 5 - 1: 4
```

Conclusão

Bibliografia

- <https://elixir-lang.org>
- <https://pragprog.com/book/elixir13/programming-elixir-1-3>
- <https://elixirschool.com/pt/lessons/basics/pipe-operator/>
- <https://speakerdeck.com/volcov/elixir-quem-e-este-pokemon>
- http://courseware.codeschool.com/try_elixir/slides/CodeSchool-TryElixir.pdf
- <https://dockyard.com/blog/2016/08/16/The-minimum-knowledge-you-need-to-start-metaprogramming-in-Elixir> http://theerlangelist.com/article/macros_1